ECE 375 Lab 3

Introduction to AVR Simulation with Atmel Studio

Lab Time: Friday 4 - 6

Faaiq Waqar

Jordan Brown

Introduction

The purpose of the third lab in ECE375 is to introduce the usage of simulation of the ATmega128 microcontroller through the use of Atmel Studio simulation, by use of manipulation using breakpoints and the process observation windows (Processor status, I/O, and Memory) on an Assembly Language program provided known as "Lab3Sample.asm". The program manipulates data and demonstrates assembly loops that created register, IO and memory manipulation that will be observed in halts during the program created by program. This as a whole will give the student a better understanding of assembly programming and debugging, simulation tools and writing too and from memory in registers and IRAM.

Program Overview

Lab3Sample.asm is a sample ASM program meant only to be used for the purpose of simulation. It is the responsibility of the student to prepare the program for simulation by first located commented section that in all caps exclaim the lines used as a breakpoint line, where the student will set the breakpoint, which will later be used for debugging and simulation observation section of the lab. According to the program description, the program will first load four registers with certain values, and then, while the program is paused, the user will copy these initialized values into data (IRAM) memory. Finally, at the end of the program, a function is called that will perform an operation using the entered values as input.

It is the job of the student to set up the program for simulation, as well as preparation by opening observatory windows for use of processor status, I/O and Memory windows. The student will take the program, Start debugging and usage of break statements, and will begin to observe and record data from the windows discussed. This includes memory held in registers and data memory. After the first portion of the simulation, the student will manipulate input of IRAM on the basis of recorded data, finish the simulation, and complete the lab.

Breakpoint Simulation

This program was only designed to be used within a simulation, but it instructs the programmer to add breakpoints and use each of the different debugging menus. These windows showed us the contents and status of all I/O registers and regular registers. Atmel also allows the programmer to see the status of the process as well as the flags that were set after any calculations. We can also use the debugger to see the flash memory that holds the program. With this windows we can view and insert 8 bit hexadecimal values into any memory address we want.

Inserting Values Into Memory

Values found in registers from earlier parts are placed into memory addresses specified by the lab handout. Pausing the program and stepping line by line adds the option to insert a value into memory. The memory window has search features to find a particular address, and clicking the values at the row allows a new one to be entered.

Function Simulation

After halting the simulation in order to map certain values to the data memory through the memory window in IRAM, we continue the solution by resuming stepping through with the debugging tool. A FUNCTION subroutine has been identified in the ASM program. We use the step into move in order to move into this subroutine, where we observe the manipulation of the stack pointer, which is important as the stack pointer is manipulated when using subroutine calls, as it is a way to use memory to jump back to the appropriate line, here we found the value to be 0X10FD, which is the address it points to. From here we stepped out of the function, and checked for the data manipulation at two specific addresses $0105:$0104.

Additional Questions

Study Questions

1. What is the initial value of DDRB?

   The initial value of DDRB was 0x00

2. What is the initial value of PORTB?

   The initial value of PORTB was 0x00

3. Based on the initial values of DDRB and PORTB, what is Port B's default I/O configuration?

   The default configuration appears to be entirely input based, due to the status of DDRB, with no pin output as a result indicated by the PORTB default

4. What 16-bit address (in hexadecimal) is the stack pointer initialized to?

   The stack pointer is pointing to the 16-bit address 0X10FF

5. What are the contents of register r0 after it is initialized?

   The contents of register R0 after initialization is 0xFF

6. How many times did the code inside of LOOP end up running?

   The code in the loop iterated 4 times

7. Which instruction would you modify if you wanted to change the number of times that the loop runs?

   In order to make this change, one may change the LDI (Load Immediate) command from $04 (The current immediate value) to the specified number of loops that you would to iterate (i.e $02 would change the loop to 2 iterations).

8. What are the contents of register r1 after it is initialized?

   The contents of initialized register R1 is 0xAA

9. What are the contents of register r2 after it is initialized?

   The contents of initialized register R2 is 0x0F

10. What are the contents of register r3 after it is initialized?

    The contents of initialized register R3 is 0x0F

11. What is the value of the stack pointer when the program execution is inside the FUNCTION subroutine?

    The value of the stack pointer in the FUNCTION subroutine s 16 bit address 0x10FD

12. What is the final result of FUNCTION?

    Contents in location: $0105 -> ba , $0104 -> 0e

Challenge

The FUNCTION subroutine featured in the sample code accepts two 16-bit values as parameters, and also returns a 16-bit value as its result. To complete the challenge for this lab, provide detailed answers to the following questions:

1. What type of operation does the FUNCTION subroutine perform on its two 16-bit inputs? How can you tell? Give a detailed description of the operation being performed by the FUNCTION subroutine.

   Per the instruction set, the 16-bit inputs are loaded sectionally onto the addressing registers, indicated by X,Y, and Z. Specifically, these will become loaded as immediate onto high and low register points for each, as in order to store the 16 bit addresses it must store respective 8 bit portions into the high and low portions. Now, as for the way the inputs are used is that they are stored into the address registered, and then referenced when doing an add with carry operation when storing the bits into the registers set at the top of the program. Arithmetic is performed on the dereferenced values, and then stored back into the address, where a branch might be used if there was a carry bit involved

2. Currently, the two 16-bit inputs used in the sample code cause the "brcc EXIT" branch to be taken. Come up with two 16-bit values that would cause the branch NOT to be taken, therefore causing the "st Z, XH" instruction to be executed before the subroutine returns.

   16 bit values 0x0000 and 0x0000 would not cause this branch to execute

3. What is the purpose of the conditionally-executed instruction "st Z, XH"?

   In the case that is the carry in the addition with carry operation, the value of X high will not move into the Z address. This is branched if this is the case after the arithmetic operation.

Difficulties

Our only difficulties came from navigating the different debugging windows, which we associated with it being our first time using this feature. Changing the values held at different memory addresses wasn't very intuitive and took us more time than expected. The rest of the windows displayed the information in a way that was easy to understand.

Conclusion

This lab was very straightforward and simple, but provided great practice using the different debugging features Atmel offers. The ability to simulate the assembly code offers the user the ability to view the exact memory addresses that they are using and diagnose the cause of problems. This is very reminiscent of ModelSim, and is similarly helpful because we can view exact values variables hold and where things are pointing. Stepping through individual lines and tracking critical values gives the programmer the best view of what is happening. Although this lab was simple, getting practice with these tools will be helpful in the future labs.

Source Code

```
;***********************************************************

;*

;*      Lab3Sample.asm

;*

;*      This is a sample ASM program, meant to be run only via

;*      simulation. First, four registers are loaded with certain

;*      values. Then, while the simulation is paused, the user

;*      must copy these values into the data memory. Finally, a

;*      function is called, which performs an operation, using

;*      the previously-entered values in memory as input.

;*

;***********************************************************

;*

;*       Author: Taylor Johnson

;*          Date: January 15th, 2016

;*

;***********************************************************


.include "m128def.inc"                          ; Include definition file


;***********************************************************

;*      Internal Register Definitions and Constants

;***********************************************************


.def   mpr = r16
```

```
.def    i = r17
.def    A = r18
.def    B = r19

;**********************************************************
;*      Start of Code Segment
;**********************************************************
.cseg                                                   ; Beginning of code segment

;**********************************************************
;*      Interrupt Vectors
;**********************************************************
.org    $0000                                   ; Beginning of IVs
                rjmp    INIT                    ; Reset interrupt

.org    $0046                                   ; End of Interrupt Vectors

;**********************************************************
;*      Program Initialization
;**********************************************************
INIT:                                                   ; The initialization routine
                ldi             mpr, low(RAMEND)        ; initialize Stack Pointer
                out             SPL, mpr
                ldi             mpr, high(RAMEND)
                out             SPH, mpr

;**********************************************************
;*      Main Program
;**********************************************************
MAIN:
```

```
              clr          r0                                  ; *** SET BREAKPOINT HERE ***
(#1)

              dec          r0                                  ; initialize r0 value



              clr          r1                                  ; *** SET BREAKPOINT HERE ***
(#2)

              ldi          i, $04

LOOP:  lsl          r1                                  ; initialize r1 value

              inc          r1

              lsl          r1

              dec          i

              brne    LOOP                          ; *** SET BREAKPOINT HERE *** (#3)



              clr          r2                                  ; *** SET BREAKPOINT HERE ***
(#4)

              ldi          i, $0F

LOOP2: inc          r2                                  ; initialize r2 value

              cp           r2, i

              brne    LOOP2                         ; *** SET BREAKPOINT HERE *** (#5)


                                                               ; initialize r3 value

              mov          r3, r2                       ; *** SET BREAKPOINT HERE *** (#6)


              ;            Note: At this point, you need to enter several values

              ;            directly into the Data Memory. FUNCTION is written to

              ;            expect memory locations $0101:$0100 and $0103:$0102

              ;            to represent two 16-bit operands.

              ;

              ;            So at this point, the contents of r0, r1, r2, and r3
              ;              MUST be manually typed into Data Memory locations

              ;              $0100, $0101, $0102, and $0103 respectively.


                                                       ; call FUNCTION

              rcall   FUNCTION                    ; *** SET BREAKPOINT HERE *** (#7)


                                                       ;  infinite  loop  at  end  of
MAIN

 DONE: rjmp   DONE


;***********************************************************

;*     Functions and Subroutines

;***********************************************************


;-----------------------------------------------------------

; Func: FUNCTION

; Desc: ???

;-----------------------------------------------------------

FUNCTION:

              ldi            XL, $00

              ldi            XH, $01

              ldi            YL, $02

              ldi            YH, $01

              ldi            ZL, $04

              ldi            ZH, $01

              ld             A, X+

              ld             B, Y+

              add            B, A

              st             Z+, B

              ld             A, X
```

```
                ld              B, Y

                adc             B, A

                st              Z+, B

                brcc    EXIT

                st              Z, XH

EXIT:

                ret                                                 ; return from rcall
```